Научная статья
УДК:004.457
doi:10.37614/2949.1215.2024.15.3.008

ОРГАНИЗАЦИЯ УПРАВЛЕНИЯ ПРОГРАММНЫМИ КОМПОНЕНТАМИ В СИСТЕМЕ КОНЦЕПТУАЛЬНОГО МОДЕЛИРОВАНИЯ С ИСПОЛЬЗОВАНИЕМ PYTHON И PYQT5

Никита Николаевич Руденко[1], Никита Александрович Вдовиченко[2]
[1, 2]Институт информатики и математического моделирования имени В. А. Путилова Кольского научного центра Российской академии наук, Апатиты, Россия
[1]nikirudi1988@mail.ru, https://orcid.org/0000-0003-0061-0397
[2]claysthree@yandex.ru, https://orcid.org/0000-0002-8150-0773

Аннотация

Данная разработка системы концептуального моделирования (СКМ) на основе PyQt5 является актуальной для визуализации и проектирования сложных систем, что важно в таких областях, как инженерия, программирование. Визуальные модели позволяют пользователям эффективно взаимодействовать с объектами, упрощая процесс построения и анализа сложных взаимосвязей. Такая СКМ полезна в образовательных целях, а также для прототипирования и демонстрации концепций наглядным образом. Возможность создавать, связывать и изменять объекты интерактивно повышает гибкость и удобство работы с концептуальными моделями.

Ключевые слова:

СКМ, визуальные модели, проектирование

Для цитирования:

Руденко Н. Н., Вдовиченко Н. А. Организация управления программными компонентами в системе концептуального моделирования с использованием Python и PyQt5 // Труды Кольского научного центра РАН. Серия: Технические науки. 2024. Т. 15, No 5. С. 97–104. doi:10.37614/2949.1215.2024.15.3.008.

Original article

ORGANIZING SOFTWARE COMPONENTS MANAGEMENT IN A CONCEPTUAL MODELING SYSTEM USING PYTHON AND PYQT5

Nikita N. Rudenko[1], Nikita A. Vdovichenko[2]
[1, 2]V. A. Putilov Institute of Informatics and Mathematical Modeling Kola Scientific Center of the Russian Academy of Sciences, Apatity, Russia
[1]nikirudi1988@mail.ru, https://orcid.org/0000-0003-0061-0397
[2]claysthree@yandex.ru, https://orcid.org/0000-0002-8150-0773

Abstract

This development of a conceptual modeling system based on PyQt5 is relevant for visualization and design of complex systems, which is important in such areas as engineering, programming. Visual models allow users to interact effectively with objects, simplifying the process of constructing and analyzing complex relationships. Such SCM is useful for educational purposes, as well as for prototyping and demonstrating concepts in a visual way. The ability to create, link, and modify objects interactively increases the flexibility and convenience of working with conceptual models.

Keywords:

SCM, visual models, prototyping

For citation:

Rudenko N. N., Vdovichenko N. A. Organization of software component management in the conceptual modeling system using Python and PyQt5 // Proceedings of the Kola Science Center of the Russian Academy of Sciences. Series: Technical Sciences. 2024. Vol. 15, No. 3. Pp. 97–104. doi: 10.37614/2949-1215.2024.15.3.008.

Введение

Система концептуального моделирования — это программа или система, которая позволяет моделировать связи между объектами или процессами в определенной предметной области. Она предназначена для формализации сложных систем и процессов в виде декларативных концептуальных моделей [1], которые помогают понять, спроектировать или проанализировать работу этих систем. Чтобы обеспечить доступ к базе данных модели в системе концептуального моделирования

с использованием Python и PyQt5, необходимо разработать архитектуру, которая связывает пользовательский интерфейс [2] с базой данных, где хранятся данные о моделях [3]. Эта архитектура должна включать несколько компонентов: интерфейс, слой доступа к данным, логику обработки данных и саму базу данных.

Перечень инструментов, предоставляемых этой системой моделирования:

— создание и редактирование концептуальных моделей. Пользователь должен иметь возможность создавать новые модели и редактировать уже существующие;

— просмотр и визуализация моделей. Интерфейс должен отображать модели в понятной форме, включая визуальные представления связей и объектов;

— обработка данных моделей. Система должна выполнять операции с данными, такие как валидация, обработка зависимостей, выполнение сценариев и анализ;

— хранение и управление данными. Модели и их данные должны храниться в базе данных для обеспечения надежного доступа, редактирования и сохранности;

— интерактивный доступ к базе данных. Пользователю требуется интерфейс для удобного взаимодействия с базой данных для создания, редактирования и поиска объектов;

В СКМ представлены следующие основные визуальные компоненты:

— формы ввода (QLineEdit, QTextEdit, QComboBox): используются для ввода информации о моделях (название модели, описание, атрибуты и т.д.);

— таблицы и списки (QTableWidget, QListView): отображение списка доступных моделей и их свойств в табличной форме;

— диаграммы и графы (QGraphicsView, QGraphicsScene): для визуализации и редактирования графических моделей, их связей и взаимодействий. Это особенно важно для отображения концептуальных моделей в виде графов;

— кнопки и панели инструментов (QPushButton, QToolBar): используются для выполнения действий: создание новой модели, редактирование, удаление, сохранение и т. д.;

— диалоговые окна (QDialog, QFileDialog): для выполнения специфических задач, таких как сохранение файла, выбор модели для загрузки, подтверждение действия и т. д.;

— графические редакторы (QCanvas, QPainter): для создания и редактирования графических объектов, таких как элементы модели (объекты, связи между ними).

К основным невизуальным компонентам системы относятся следующие:

— слой доступа к данным (Data Access Layer, DAL): отвечает за взаимодействие с базой данных, управление запросами на получение, изменение, удаление и добавление данных. DAL изолирует приложение от конкретной реализации базы данных (например, SQL или NoSQL) и предоставляет интерфейс для работы с данными;

— логика управления моделями (Business Logic Layer, BLL): компонент управляет логикой взаимодействия с моделями и их атрибутами. Он обрабатывает бизнес-правила, осуществляет валидацию данных, контролирует корректность связей между моделями и выполняет вычисления, связанные с моделями;

— контроллер взаимодействия с пользователем (Controller/Presenter): управляет связью между пользовательским интерфейсом (визуальными компонентами) и бизнес-логикой системы. Контроллер принимает действия пользователя, передает данные в слой бизнес-логики, а затем обновляет интерфейс на основе полученных данных;

— модуль взаимодействия с внешними системами (API, интеграция): если система концептуального моделирования должна взаимодействовать с внешними системами или обмениваться данными через API (например, для интеграции с другими инструментами моделирования), этот компонент будет отвечать за такие операции;

— модуль валидации данных (Data Validation): компонент проверяет правильность данных,вводимых пользователем или получаемых из внешних источников, перед их сохранением в базу данных.

В ходе разработки системы концептуального моделирования было принято решение использовать язык программирования Python и библиотеку PyQt5 для реализации как визуальных, так и невизуальных компонентов. Это решение было обосновано следующими причинами:

- простота и гибкость Python:
  - является высокоуровневым языком программирования, известным своей простотой и читаемостью кода. Это позволило ускорить процесс разработки, сосредоточив внимание на реализации логики системы, а не на сложных технических деталях;
  - широкий выбор библиотек и фреймворков для различных задач (работа с базами данных, API, валидация данных и т. д.) сделал Python идеальным инструментом для создания гибкой и расширяемой архитектуры системы;
- PyQt5 как мощный инструмент для создания графических интерфейсов:
  - предоставляет богатый набор готовых визуальных компонентов (кнопки, формы, таблицы, графические виджеты и др.), которые позволяют быстро создавать функциональные и удобные пользовательские интерфейсы;
  - поддерживает возможность создания сложных графических интерфейсов с использованием визуальных элементов для работы с графами, диаграммами и другими компонентами, необходимыми для визуализации концептуальных моделей;
  - кроссплатформенность PyQt5 дает возможность разработанной системе работать на различных операционных системах (Windows, macOS, Linux) без необходимости внесения значительных изменений в код;
  - интеграция с базами данных и внешними сервисами;
  - Python предоставляет простой и удобный способ взаимодействия с различными базами данных (SQL, NoSQL) через библиотеки, такие как SQLite, SQLAlchemy и др., что сделало возможным быстрое и удобное внедрение слоя доступа к данным (DAL);
  - PyQt5 также хорошо сочетается с библиотеками для работы с сетевыми запросами и API (например, requests), что упростило интеграцию системы с внешними сервисами и модулями;
  - быстрое прототипирование и тестирование;
  - благодаря своей интерпретируемой природе Python позволяет быстро прототипировать и тестировать различные решения, что сделало его идеальным выбором для гибкой разработки и итерационного процесса создания системы.

Связь СКМ и ГИС

Связь СКМ с геоинформационными системами (ГИС) [4] открывает новые возможности для анализа пространственных данных и их визуализации. В рамках такой интеграции СКМ может использоваться для создания моделей объектов, которые имеют пространственную привязку, например, инфраструктурных элементов, транспортных сетей, зданий и природных объектов. В ГИС данные этих объектов могут быть дополнены их географическим положением, координатами и пространственными характеристиками. Таким образом, концептуальные модели могут быть отображены на карте и связаны с реальными геоданными, что позволит визуализировать их влияние в контексте реальной местности.

Это объединение может использоваться, например, для моделирования логистических или транспортных систем, где важно учитывать географическое расположение элементов. СКМ позволит легко моделировать отношения между объектами (дороги, станции, маршруты), а ГИС предоставит платформу для анализа пространственных аспектов, таких как расстояния, топология или доступность ресурсов.

Использование сокетов (Sockets) для взаимодействия между распределенными компонентами

Python поддерживает взаимодействие с использованием сокетов через стандартную библиотеку socket, которая предоставляет все необходимые инструменты для реализации сетевого обмена

данными с использованием протоколов TCP/IP и UDP. Этот подход часто используется для организации взаимодействия между компонентами, расположенными на разных серверах или устройствах.

Что такое сокеты: сокет — это конечная точка в сетевом соединении, через которую компоненты системы могут отправлять и получать данные. Компоненты могут обмениваться сообщениями или передавать данные через сеть, используя сокеты.

Пример использования сокетов в Python: один из компонентов может выступать в роли сервера, который слушает определенный порт и обрабатывает запросы, поступающие от других компонентов (клиентов). Клиенты подключаются к серверу по указанному адресу и обмениваются данными через сокеты.

Основной идеей разработки системы является использование открытой архитектуры, что позволяет расширять функциональность комплекса, добавляя новые визуальные и невизуальные компоненты, а также компоненты, взаимодействующие с внешними приложениями через стандартные протоколы.

- Menu Supervisor: в системе предусмотрен центральный компонент Menu Supervisor, который управляет компонентами системы и координирует их взаимодействие. Этот компонент отвечает за:
  — мониторинг изменений в данных, с которыми работают компоненты (в том числе внешние приложения);
  — вызов и управление опциональными компонентами, которые могут быть активированы при изменении данных;
  — управление потоком данных между компонентами, обеспечивая динамическое обновление визуальных элементов при изменении данных в системе.
- ExecAgents: для обеспечения запуска компонентов на удаленных узлах используются агенты выполнения (ExecAgents). Каждый агент отвечает за запуск тех или иных компонентов на локальном устройстве, с учетом переданных параметров таких, как:
  — IP-адрес устройства, на котором должен быть запущен компонент;
  — путь к компоненту в файловой системе.

ExecAgents запускаются на удаленных компьютерах и обеспечивают вызов необходимых компонентов через сети. Коммуникация между агентами и Menu Supervisor происходит через TCP/IP протоколы с использованием стандартных механизмов обмена данными (например, через sockets или Pyro4 для удаленных вызовов процедур).

Компонент DataAccess отвечает за доступ к данным и информирование Menu Supervisor о любых изменениях в данных, с которыми работают компоненты. DataAccess периодически передает обновления об изменении данных через сообщения, что позволяет системе автоматически обновлять те компоненты, которые зависят от этих данных.

Компоненты (визуальные и опциональные): визуальные компоненты (разработанные на PyQt5) представляют собой интерфейсы, которые отображают данные в системе. Эти компоненты могут включать в себя различные типы данных, такие как геоинформационные данные или результаты имитационного моделирования. Опциональные компоненты могут быть запущены по запросу в зависимости от типов данных, с которыми работает система, или изменений в этих данных. Эти компоненты выступают в роли индикаторов изменений и могут быть динамически подключены или отключены в зависимости от конфигурации проекта.

Процесс добавления компонентов

На начальной стадии работы с проектом оператор указывает, какие компоненты должны быть активированы в системе. Это включает:

— определение списка визуальных компонентов, которые будут отображать данные пользователю;

— задание типов данных, с которыми эти компоненты работают (например, геоинформационные данные, данные симуляций и т.д.);

— указание идентификаторов для каждого компонента и его соответствие типам данных.

Взаимодействие компонентов

Menu Supervisor обеспечивает информационно-управляющие потоки данных между компонентами через:

— TCP/IP протоколы, что позволяет компонентам взаимодействовать даже при их размещении на удаленных машинах;

— Sockets или Pyro4, которые могут использоваться для организации обмена данными между распределенными компонентами системы.

Каждый визуальный компонент или компонент данных уведомляется о любых изменениях в данных через Menu Supervisor, который получает информацию об изменениях от компонента DataAccess. Это позволяет поддерживать актуальность данных во всех частях системы и автоматически обновлять интерфейсы.

Гибкость и динамичность системы

Одной из ключевых особенностей системы является возможность опционального добавления или исключения компонентов. На начальной стадии работы с проектом оператор может определить, какие компоненты необходимы, а какие можно исключить. Это позволяет легко адаптировать систему под текущие задачи, без необходимости вносить серьезные изменения в архитектуру программного комплекса.

В отличие от технологий DCOM и OLE, подход, реализованный на Python с использованием TCP/IP, не требует дополнительной настройки клиентов и является кроссплатформенным. Это позволяет компонентам взаимодействовать через Интернет или локальные сети с высокой степенью надежности передачи данных.

Пример использования сокетов для взаимодействия компонентов

Для организации информационно-управляющих потоков данных между компонентами можно использовать Python-библиотеку socket. Например:

Листинг 1

Сервер для Menu Supervisor

```
import socket

def start_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('localhost', 8080))
    server_socket.listen(5)
    print("Menu Supervisor запущен и ожидает соединений...")

    while True:
        client_socket, addr = server_socket.accept()
        print(f"Соединение установлено с {addr}")
        data = client_socket.recv(1024).decode()
        print(f"Получены данные: {data}")
        client_socket.send("Данные получены".encode())
        client_socket.close()

if __name__ == "__main__":
    start_server()
```

Листинг 2

Клиент для ExecAgent

```
import socket

def send_data():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('localhost', 8080))
    client_socket.send("Запуск компонента".encode())
    response = client_socket.recv(1024).decode()
    print(f"Ответ от Menu Supervisor: {response}")
    client_socket.close()

if __name__ == "__main__":
    send_data()
```

Технология OLE Automation и аналогичные подходы в Python

Рассмотрение OLE Automation параллельно альтернативами на языке Python важно для автоматизации взаимодействия с разными операционными системами. OLE Automation позволяет управлять этими программами из других приложений, что полезно для задач, связанных с интеграцией и обменом данными между системами. Однако Python предлагает другие подходы, такие как REST API, и других сетевых технологий для межпроцессного взаимодействия, что может быть более гибким и кроссплатформенным решением. Таким образом, рассмотрение альтернатив помогает выбрать наиболее подходящий метод для конкретной задачи.

В экосистеме Python нет точного аналога OLE Automation из Windows, однако схожие функции можно реализовать с помощью технологий удаленного вызова процедур (Remote Procedure Call, RPC) или через использование библиотек для межпроцессного взаимодействия (Inter-Process Communication, IPC). В Python есть несколько способов для обеспечения такого взаимодействия между компонентами.

Использование библиотеки xmlrpc для RPC

Технология XML-RPC позволяет компонентам взаимодействовать друг с другом по сети, вызывая удаленные методы. Один из компонентов системы может быть сервером, который предоставляет определенные функции (методы), а другой компонент может выступать в роли клиента, который вызывает эти методы удаленно. Пример простого XML-RPC-сервера и клиента:

Листинг 3

XML-RPC сервер:

```
from xmlrpc.server import SimpleXMLRPCServer

def add_numbers(x, y):
    return x + y

server = SimpleXMLRPCServer(('localhost', 8000))
print("Сервер запущен, ожидает вызовов...")
server.register_function(add_numbers, 'add')
server.serve_forever()
```

Листинг 4

XML-RPC клиент:

```
import xmlrpc.client

client = xmlrpc.client.ServerProxy('http://localhost:8000')
result = client.add(5, 3)
print(f"Результат вызова удаленной функции: {result}")
```

В этом примере клиент вызывает удаленную функцию add на сервере, передавая параметры, а сервер выполняет сложение и возвращает результат клиенту (Листинг 3,4).

Модуль Pyro4 для распределенных компонентов

Pyro4 — это библиотека, которая реализует механизм удаленного вызова процедур, позволяя компонентам системы взаимодействовать друг с другом по сети. Pyro4 позволяет определить один компонент как сервер (который предоставляет удаленные методы), а другой как клиент (который вызывает эти методы). Пример Pyro4:

Листинг 5

Pyro4 сервер:

```
import Pyro4

@Pyro4.expose
class ModelManager:
    def get_model_data(self, model_id):
        return f"Данные модели {model_id}"

def start_server():
    Pyro4.Daemon.serveSimple(
        {
            ModelManager: "model.manager"
        },
        ns = False)

if __name__ == "__main__":
    start_server()
```

Pyro4 позволяет создать распределенную систему, где один компонент (сервер) предоставляет данные или методы для управления моделями, а другой компонент (клиент) обращается к этим методам удаленно (Листинг 5, 6).

Листинг 6

Pyro4 клиент:

```
import Pyro4

def fetch_model_data(model_id):
    model_manager = Pyro4.Proxy("PYRO:model.manager@localhost:9090")
    return model_manager.get_model_data(model_id)

if __name__ == "__main__":
    data = fetch_model_data(1)
    print(f"Полученные данные: {data}")
```

Заключение

В результате проведенного анализа была предложена модульная архитектура программного комплекса для системы концептуального моделирования, основанная на использовании языка программирования Python и библиотеки PyQt5. Эта архитектура предоставляет несколько возможностей реализации, каждая из которых может быть адаптирована в зависимости от специфики задач и требований пользователей.

Данная система имеет потенциал для дальнейшего развития для автоматизации анализа данных и расширение функциональности за счет подключения дополнительных модулей и API.

Такие направления могут значительно повысить ценность программного комплекса и его конкурентоспособность. В дальнейшем планируется внедрить в СКМ методы ГИС, которые позволят расширить функциональность системы концептуального моделирования, добавив анализ пространственных данных и интеграцию с реальными картографическими объектами. Это даст возможность моделировать процессы с учетом географических факторов, улучшая точность планирования и анализа в таких сферах, как логистика, городское планирование и управление ресурсами.

Список источников

1. Павлецов А., Фридман А. Организация управления программными компонентами в системе концептуального моделирования // Информационные технологии в региональном развитии. Апатиты, 2003. Вып. III. С. 63–68.
2. Матченко И., Павлецов А., Фридман А., Фридман О. Основы пользовательских и внутренних интерфейсов системы ситуационного моделирования // Математические методы описания и исследования сложных систем. Апатиты: КНЦ РАН, 2001. С. 119–132.
3. Матченко И., Фридман А. Компонент доступа к данным для пакета ситуационного концептуального моделирования // Информационные технологии в региональном развитии. Апатиты, 2003. Вып. III. С. 86–93.
4. Фридман А., Курбанов В. Формальная концептуальная модель промышленно-природного комплекса как инструмент управления вычислительными экспериментами // Труды СПИИРАН. 2014. № 6(37). С. 424–453.

References

1. Pavletsov A., Fridman A. Organization of software component management in a conceptual modeling system // Information Technologies in Regional Development. Apatity, 2003. Issue III. P. 63–68.
2. Matchenko I., Pavletsov A., Fridman A., Fridman O. Fundamentals of user and internal interfaces of the situational modeling system // Mathematical Methods for Describing and Studying Complex Systems. Apatity: KSC RAS, 2001. P. 119–132.
3. Matchenko I., Fridman A. Data access component for the situational conceptual modeling package // Information Technologies in Regional Development. Apatity, 2003. Issue III. P. 86–93.
4. Fridman A., Kurbanov V. Formal conceptual model of an industrial-natural complex as a tool for managing computational experiments // Proceedings of SPIIRAS. 2014. No. 6(37). P. 424–453.

Информация об авторах

Н. Н. Руденко — студент 1 курса аспирантуры;
Н. А. Вдовиченко — студент 1 курса аспирантуры.

Information about the authors

N. N. Rudenko — 1st year student of the Postgraduate School of the Kola Science Center of the Russian Academy of Sciences;
N. A. Vdovichenko — 1st year student of the Postgraduate School of the Kola Science Center of the Russian Academy of Sciences.

Статья поступила в редакцию 01.10.2024; одобрена после рецензирования 10.10.2024; принята к публикации 21.10.2024.
The article was submitted 01.10.2024; approved after reviewing 10.10.2024; accepted for publication 21.10.2024.